CS 553 Cloud Computing
Programming Assignment 2
Archi Dsouzal (CWID: A20375329)
Piyush Nath (CWID : A20380531)
Report

Problem statement:-

This programming assignment covers the TeraSort application implemented in 3 different ways: Java, Hadoop, and Spark. Your sorting application could read a large file and sort it in place (your program should be able to sort larger than memory files, also known as external sort). You will create 2 datasets, a small and a large dataset, which you will use to benchmark the 3 approaches to sorting: 128GB dataset and 1TB dataset. You must generate your datasets using the file generator at since storing 1TB dataset on Amazon S3 would be both expensive and slow, you are encouraged to generate the 1TB dataset on demand every time you want to benchmark your system.

Overview Of Software Packages used

Linux version:- Ubuntu 14.04.4 LTS
Java Version:- java version "1.7.0_99" and java-8-oracle
OpenJDK Runtime Environment (amzn-2.6.5.0.66.amzn1-x86_64 u99-b00)
Hadoop Version:- hadoop-2.7.2
Spark Version:- spark-1.6.0
Open MPI

Methodology:-

Shared Memory :

Algoirthm

1) Initialize the thread pool of based on size passed.
2) Decides the byte processed by each thraed.
3) Intiliaize the SortFile object with all data.
4) Add job to thread pool to execute.
5) SortFile worker thread will decides what job he needs to do SORT+MERGE or MERGE
6) IF SORT+MERGE job it first calculate total pages it needs to process based on PAZE_SIZE
7) Using loop it reads data of PAGE_SIZE, sort it and creates the new file and writes data to new file and save the file name in
filelist.
8) Once it sort the all the pages and it pass the filelist to the merge function.
9) Merge merge sorted filelist in K-way merge
10) Once all thread complete the SORT+K_MERGE it deletes the old file and notify the main thread.
11) Main Thraed then look into sort dir and create the list of files
12) And calls the SortFile merge function using worker thread.
13) It merger the data same way as earlier.

Sorted file created in SORTJOB directory. It will follow below convention
SORT_<THREAD_COUNT>MERGE

Instruction to run shared memory sort :

$sudo apt-get install default-jdk
$mkdir shm
$export CLASSPATH = $CLASSPATH:$HOME/shm
$ar cvfe kwayExternalSort.jar TaskPool *.class

Copy the gensort,and valsort, shm_run.sh to one dir Run below command
It will generate data using gensort and sort and validate using valsort

$./shm_run.sh

Software version used for Hadoop and spark installations:-

Linux version:- Ubuntu Server 16.04 LTS (HVM), SSD Volume Type
Java Version:- version 1.8.0_151 and java-8-oracle
Hadoop Version:- hadoop-2.8.2
Spark Version:- spark-2.2.0

Hadoop

1) core-site.xml:-

It has the details for namenode demon we also keep the details for the temp folder, which is utilized by the hadoop system during map and reduce. Here we have to give the namenode address which drives.

2) hdfs-site.xml:-

The hdfs-site.xml file contains the configuration settings for HDFS daemons; the NameNode, theSecondary NameNode, and the DataNodes. Here, we can configure hdfs-site.xml to specify default block replication and permission checking on HDFS. The actual number of replications can also be specified when the file is created. HDFS-SITE we also set the number of replication and also we can set the size of datablock so that number of itteration can be reduced.

3) mapred-site.xml:-

It has the details to keep tracker for jobs running at the instance.

4) slaves:-

This is the list of slave nodes, which is used to create datanode and tracking node.

5) master:-

The Masters contain a list of hosts, one per line, that are required to host secondary NameNode servers. The Masters file informs about the Secondary NameNode location to Hadoop daemon.

6) hadoop-env.sh:-

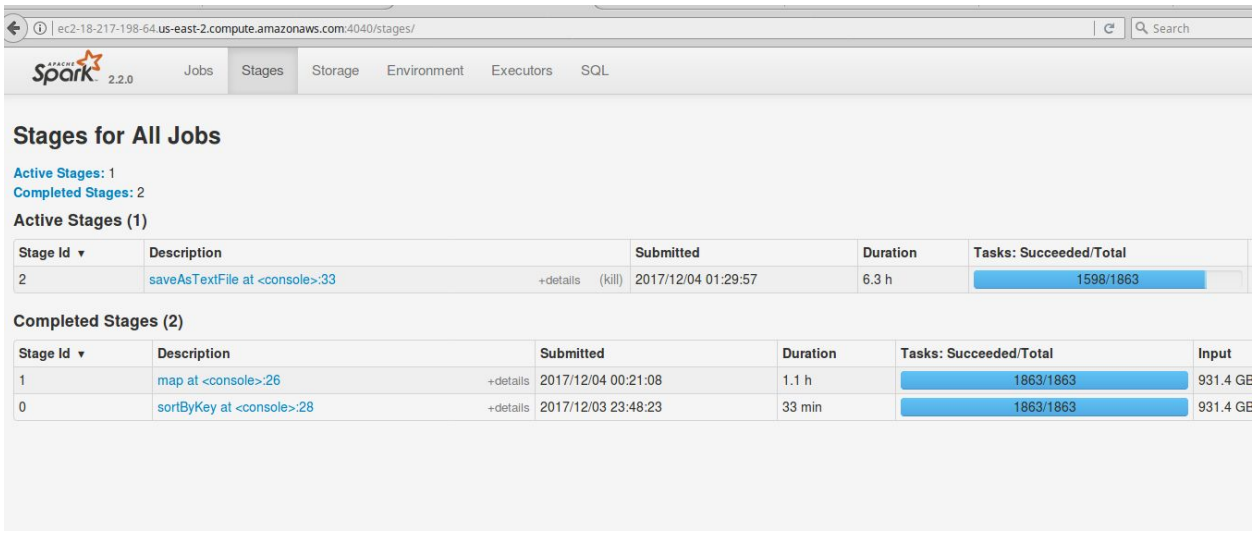
In this we set JAVA_HOME variable with java path.

When we are doing multi node ,we have to make sure that core and hdfs-site.xml is set others are not touched.

Important point, when we added EBS volume to work on our instance. It was observed that when we are using our system was behaving very slow. Specially when we are working on the EBS volume. This could be one of the reason that we have observe some slow response from the system.

Spark

Mark has its own page because, while it can run in Hadoop clusters through YARN (Yet Another Resource Negotiator), it also has a standalone mode. The fact that it can run as a Hadoop module and as a standalone solution makes it tricky to directly compare and contrast. However, as time goes on, some big data scientists expect Spark to diverge and perhaps replace Hadoop, especially in instances where faster access to processed data is critical spark.
Spark is very well known for its ease to use in simple scala . Like in our experiment , where we had to sort 1Tb of data. It was important to divide the data to key and value. So that high intensive comparison can only be done on the key part of the data . And it will save unnecessary loss of the computation to be happening.



In our case we are dividing our task into three mode:-

1) Sort by key value.
2) Map
3) Saving to file (high intensive)

But as spark is very fast compared to hadoop is because its a in memory processing delivery to real time analytics where as map and reduce is strict disk dependent . since when our system has so much of high processing speed, latency for memory becomes bottleneck. And we have to work on it to improve the computation.

Steps and challenges:

We started with exploring the AWS. where every click opens something new and It was exciting yet thrilling for exploration. This assignment helped us to get in details about it. It was fascinating to see how daemons were managing files and they were distributed among other datanodes, and it was managed by the namenode.

We observed that as we increase the number of core, there was a good improvement in the performance in spark in comparison with hadoop. But when we were running our instance in low configuration. It looked like our external sort was the one which performed best among others.

When we were running code for huge data, Program was taking hours to process it.

hdfs balancer -threshold 5 we use this to make sure that cluster is balanced.

It is important to keep in mind when running haoop or spark in multiple cluster is to format each node before starting the service from the master node. Otherwise we wont be able to add desired files to our DFS.

Due to less credit and limitation in time , we had reduced the Amount of file to be processed for multi node cluster to 100.

Performance Table

Experiment (instance/dataset)	**Shared Memory**	**Hadoop**	**Spark**	**MPI**
Compute Time (sec) [1xi3.large 128GB]	5760 Sec	4hrs, 36mins, 53sec =16613	2hr 8mins=768 0 sec	2 GB Local 244 Second
Data Read (GB) [1xi3.large 128GB]	384 GB	1280000000	66.1 GB	6 GB
Data Write (GB) [1xi3.large 128GB]	384 GB	1280000000	66.1 GB	6 GB
I/O Throughput (MB/sec) [1xi3.large	136 MB/sec	15.4096 MBps	17.21MBps	49 MB \Sec

128GB]				
Compute Time (sec) [1xi3.4xlarge 1TB]	32763 Sec	11hrs 24mins 56 sec=41096	9 hrs 3 mins= 32580	
Data Read (GB) [1xi3.4xlarge 1TB]	3072 GB	10000000000	523.5 GB	
Data Write (GB) [1xi3.4xlarge 1TB]	3072 GB	10000000000	523.5 GB	
I/O Throughput (MB/sec) [1xi3.4xlarge 1TB]	192 MB/sec	48.666 MBps	64.27 MBps	
Compute Time (sec) [8xi3.large 100GB]	N/A	2 hr 25 mins		
Data Read (GB) [8xi3.large 100GB]	N/A	1000000000		
Data Write (GB) [8xi3.large 100GB]	N/A	1000000000		
I/O Throughput (MB/sec) [8xi3.large 100GB]	N/A	46.98 MBps		
Speedup (weak scale)	1.2	3		
Efficiency (weak scale)	25%	69%		

Throughput Calculation for Shared memory : (Total read + total write) / computation time
Note: Since we run out amazon credit so we ran MPI on local
Local Node configuration : 8 Gb RAM i3 Processor(2 Core) 500 HDD

Comments on Shared memory performance analysis.

A. Compute Time (sec) [1xi3.large 128GB] : 5760 Sec

Comment : I have used the 4 thread and page size as ~50 MB. Each thread first creates the file of 50 MB and sort and letter merge small 50MB chunks using K-way merge. I have tried using the 1 thread with less page size and has very low performance. One thing is noticeable in java code i have used BufferedReader, before

that i was using RandomAccess file reader which is 10 times slower than BufferedReader. Paze size is limited by Java limitation. Java throw out of heap space error if page size is increase above the 50MB.
Performance can be further boost by using buffer for writing the data which will hold large portion of data before writing data to disk.

B. Data Read (GB) [1xi3.large 128GB] : 384 GB

Comment : In this implementation of shared memory reads every data 3 times. Two times by worker thread and one time by the worker thread. First time it is read by the worker thread to sort. Next time it read to merge during K-way merge. Main thread read for final merge. This makes that total read is 128GB X 3 = 384GB

C. Data Write (GB) [1xi3.large 128GB] : 384 GB

Comment : Like implementation data write three time when it is read. This makes the total data write 3 time the size of data. 128GB X 3 = 384GB

D. I/O Throughput (MB/sec) [1xi3.large 128GB] 136 MB/sec

Comment : Even though I can SSD has high bandwidth then the result. Java limitation cause the not not more then 50 MB data processed at time. So only 200 MB/Sec can max achieved under this implementation of shared memory in Java using 4 thread.

I/O Throughput = (total Read Write) / computation

E. Compute Time (sec) [1xi3.4xlarge 1TB] : 32760 Sec

Comment : I have used the 16 thread and page size as ~50 MB. Each thread first creates the file of 50 MB and sort and letter merge small 50MB chunks using K-way merge. I have tried using the 1 thread with less page size and has very low performance. One thing is noticeable in java code i have used BufferedReader, before that i was using RandomAccess file reader which is 10 times slower than BufferedReader. Paze size is limited by Java limitation. Java throw out of heap space error if page size is increase above the 50MB.
16 Thread is really useful to scale up the performance. Data process at single moment is 16 * 50 MB vs 4 * 50 MB. Same while merging.

F. Data Read (GB) [1xi3.4Xlarge 1TB] : 3072 GB

Comment : In this implementation of shared memory reads every data 3 times. Two times by worker thread and one time by the worker thread. First time it is read by the worker thread to sort. Next time it read to merge during K-way merge. Main thread read for final merge. This makes that total read is 1024GB X 3 = 3072GB

G. **Data Write (GB) [1xi3.4xlarge 1TB] : 3072 GB**

Comment : Like implementation data write three time when it is read. This makes the total data write 3 time the size of data. 1024GB X 3 = 3072GB

H. **I/O Throughput (MB/sec) [1xi3.4xlarge1TB] 192 MB/sec**

Comment : Even though I can SSD has high bandwidth then the result. Java limitation cause the not not more then 50 MB data processed at time. So only 200 MB/Sec can max achieved under this implementation of shared memory in Java.

I. **Speed up Weak scaling : 1.2 times**

Comment Performance is improved 1.2 times as no of resources has increased 4 times as well as data performance has improved 1.2 times.

J. **Efficiency :**

Comment : With increase in resources as well as data this implementation has achieved the 25% more efficient under previous resources.

Conclusion : Java is not best choice to implement the Terasort. It has many limitation. Increasing the page size causes the out of heap space exception. Paze to be read by limited by the maximum Integer. **Garbage collection is illusion**. Even though SSD has high memory bandwidth, it is limited by the Java limitation. Adding more core and RAM can improve performance.

MPI :

I am using the same code as shared memory with little modification. I am using native open mpi with Java support. To install MPI with java support is given in instruction. We have not tried the MPI in the Amazon EC2 instances. MPI program is run using mpi_run.sh script. This script first divides the file based on no of processes and call MPI code. MPI code pickup up respective file based on rank and perform Sorting same as shared memory.

When process rank other than zero completes the sorting it sends message to rank 0 process indicating it has done the job.

MPI.COMM_WORLD.send(message,1, MPI.INT, 0, 1);

Once the Rank 0 process receives the message from the other rank processes it gathers all the file to be merged from other processes and call merge process again.

This is single node implementation of MPI.

The performance is similar to the shared memory when locally tested.

Spark & Hadoop:

A. Compute Time (sec) [1xi3.large 128GB] : 16613 and 7680 Sec

Comment : In this implementation ,we have executed random generated code of 128GB ,in hadoop we didnt explicitly called sorting algorithm as when we assignme value to key and value.. And at time of map hadooop sort wrt its key. An

B. Data Read (GB) [1xi3.large 128GB] : 100 GB and 66.1 GB

Comment : In this implementation of data read, Hadoop system reads every data once and sort it but in spark we have concept of cache in memory which reduces the disk call.

C. Data Write (GB) [1xi3.large 128GB] : 128 GB and 66.1 GB

Comment : Same as mentioned above

D. I/O Throughput (MB/sec) [1xi3.large 128GB] 15.4 MB/sec and 17 MB/s

Comment : Even though there are lot of bandwidth there for memory disk we are able to get 15.4 MB/sec or 17 .the reason is ,here data is sent and received in terms of messages and every time a connection has to be established to get data. Which is more costly process then simply getting data in single node.

E. Compute Time (sec) [1xi3.4xlarge 1TB] : 41096 sec and 32580 sec

Comment : In this we are executing 1 TB of data ,which is approx 8 times more work ,which computing will calculate to be same time as we are also increasing the number of cores by 8. But we have observed that for 1TB it is taking thrice the time. Which is happening because number of Disk reads have increased, though we have same computation but if we observe,the number of map read has increased. This can be reduced further if we increase of block size from 256 MB to 1 GB. as we are reducing the number of itteration from ram with disk.

F. Data Read (GB) [1xi3.4Xlarge 1TB] : 1TB and 523.5GB

Comment : In this implementation of data read, Hadoop system reads every data once and sort it but in spark we have concept of cache in memory which reduces the disk call.

G. **Data Write (GB) [1xi3.4xlarge 1TB] : 3072 GB**

Comment : same as mentioned above

H. **I/O Throughput (MB/sec) [1xi3.4xlarge1TB] 48 MB/sec and 64.27 MBps**

Comment : Even though there are lot of bandwidth there for memory disk we are able to get 48 MB/sec or 64 .the reason is ,here data is sent and received in terms of messages and every time a connection has to be established to get data. Which is more costly process than simply getting data in single node. But its giving better throughput from previous configuration as there is inc

I. **Speed up Weak scaling : 3 times**

Comment :

Even thoguh we have improved the environment by 8 times ,we are getting speed up only by 3 times, as number of Disk call has also increased with it. And it is reducing the total speed up

Conclusion:-

As per the results and observation ,we can see that shared memory is faster than both hadoop and spark . as they have to setup master and slave communication to manage the work through message exchange @network.

But It was fascinating to observe that Spark is giving better performance as we are doing less input output operation with the disk. Spark is more memory intensive operations unlike hadoop being disk intensive.

Compare the performance of the three versions of TeraSort (Shared-Memory, Hadoop, Spark, and MPI [EC]) on 1 node scale on two different types of instances as well as on a virtual cluster of 8 nodes.

Ideally shared-memory's performance at one node should be better because it does not have any startup cost associated with it while in case of Hadoop and spark they have to setup master and slaves communication to manage the work through message exchange and MPI is performing better .

But at 8 nodes we observed that Hadoop is running 4 times faster then compared to what it ran for single node instance of 2 VM. Spark had better performance as number of IO operations were reduced.

And Spark is the best among all 4.as Spark's performance is better than Hadoop because it has efficient use of memory than Hadoop due to it's RDD (Resilient Distributed Datasets) architecture.

Which seems to be best at 1 node scale?

External Sort in single node,because it does not have any extra cost for startup and processing through message exchange.

How about 8 nodes?

8 Nodes Spark should be faster, because it uses RDD. and it has less Disk communication because of its in memory sort which reduces the bottleneck performance for disk latency.

which would be best at 100 node scale?

Still for 100 Nodes Spark should be faster, because it uses RDD. as more memory of data will be available to process.

How about 1000 node scales?

Still for 1000 Nodes Spark should be faster .because it will use more memory data to process.

CloudSort Benchmark?

what is the cheapest you can sort a fixed huge cloud memory (data in distributed environment) . , this benchmark will not only point out the best platforms for building data pumps, but also find the most efficient sort implementations from a total cost of ownership perspective.

Reference :

- https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/examples/terasort/package-summary.html
- http://www.oracle.com/technetwork/java/javase/downloads/index.html
- http://hadoop.apache.org/docs/current1/mapred_tutorial.html
- http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html
- http://spark.apache.org/downloads.html
- http://spark.apache.org/docs/latest/cluster-overview.html
- http://www.ordinal.com/gensort.html
- http://sortbenchmark.org
- http://sortbenchmark.org/2014_06_CloudSort_v_0_4.pdf
- http://www.novixys.com/blog/setup-apache-hadoop-cluster-aws-ec2/
- https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-common/yarn-default.xml

- https://stackoverflow.com (for quick query solves)